Reproducing Generalizing Hamiltonian Monte Carlo with Neural Networks

Samuel Laferrière
Montreal Institute for Learning Algorithms
Université de Montréal
Montréal, QC H3T 1N8
samuel.laferriere.cyr@umontreal.ca

Joey Litalien
Department of Electrical Engineering
McGill University
Montréal, QC H3A 0E9
joey.litalien@mail.mcgill.ca

Abstract

As part of the ICLR 2018 Reproducibility Challenge, we reproduce the results from Learning to Hamiltonian Monte Carlo (L2HMC; Lévy et al., 2018), a general-purpose method to train Markov chain Monte Carlo (MCMC) kernels, parameterized by deep neural networks, that converge and mix quickly to their target distribution. We reproduce the core part of the work, namely automatic training of MCMC samplers applied to simple yet challenging distributions where Hamiltonian Monte Carlo (HMC) tends to fail. We show that most experiments conducted in the paper are correct and further propose an additional test to better understand subtle design decisions made by the authors. The code for all experiments is available online on Github.[1]

1 Paper Summary

1.1 Motivation

Efficient sampling from high-dimensional distributions is among the long-standing goals of Bayesian inference. Exact posterior inference being typically intractable in these cases, one must resort to approximate statistical inference models. One family of such algorithms, Markov chain Monte Carlo (MCMC), operates by drawing a series of correlated samples that will converge in distribution to the target equilibrium distribution π. MCMC methods are asymptotically unbiased and applicable even when it is impossible to sample directly from π, making them a powerful and versatile toolbox for statisticians and machine learners alike.

This elegant framework is not without problems, however. For complex distributions with many parameters, simple variants of MCMC such as Random Walk Metropolis (Metropolis et al., 1953) and Gibbs sampling (Geman and Geman, 1984) usually fail to converge (*burn-in*) in a reasonable amount of time, thus resulting in poor exploration of the typical set and highly biased estimators. Even if the chain has burned-in, it is also difficult to determine if it has produced uncorrelated samples (*mix-in*). This is mainly due to the undesirable tendency of these models to explore parameter space via inefficient random walks.

Hamiltonian Monte Carlo (HMC; Duane et al., 1987, Neal et al., 2011) is a MCMC algorithm that is able to suppress this random walk problem and sensitivity to correlated samples by simulating Hamiltonian dynamics in an augmented space. By transforming the density function to a potential energy function and introducing auxiliary momentum variables v, HMC lifts the target distribution onto a joint probability distribution on phase space (x, v), where x is the original variable of interest seen as position. A new state is then obtained by solving the equations of motion for a fixed period of time using a volume-preserving integrator such as Leapfrog. This is better visualized in Figure 1.

[1] github.com/joeylitalien/l2hmc

Submitted as course project for IFT6135 H2018 Representation Learning

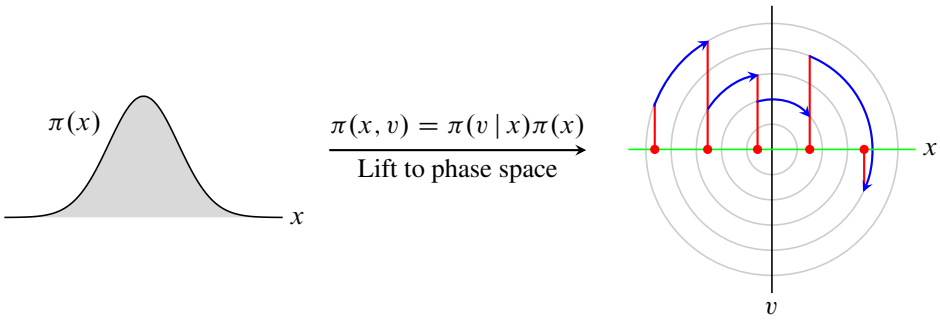

Figure 1: *Visualizing HMC for a 1D Gaussian* (Example from Betancourt, 2017). Each Hamiltonian Markov transition lifts the initial state onto a random level set of the Hamiltonian, which can then be explored with a Hamiltonian trajectory before projecting back down to the target parameter space. Pictorially, each trajectory corresponds to a frictionless particle navigating a quadratic bowl after a gentle push in either direction ($\pm\infty$).

The addition of random (usually normally distributed) momenta favors long distance jumps in state space with a single Metropolis–Hastings (MH) step. Nonetheless, HMC can still perform poorly in a number of cases such as multi-modal distributions, ill-conditioned energy landscapes, and rapidly changing gradients. HMC also requires the gradient of the log-posterior to be available, something that often necessitates computationally-expensive tools such as automatic differentiation.

Motivated by recent work on probabilistic models parameterized by deep nets, Learning to Hamiltonian Monte Carlo (L2HMC; Lévy et al., 2018) presents a learned inference architecture that generalizes HMC. The proposed algorithm learns a MCMC kernel by training a modified Leapfrog integrator parameterized by neural nets. The parameters are trained to encourage fast mixing and convergence to its target distribution. While this transition kernel is no longer symplectic, it retains strong theoretical guarantees with a tractable MH accept/reject step, making L2HMC potentially capable of sampling very challenging distributions.

1.2 Proposed approach

Update rules. Hamiltonian Monte Carlo has only two parameters: the number of jumps L and the jump size ϵ. One obvious way to generalize this method would be to use neural networks to learn both of these parameters. In ravine-like regimes (*e.g.*, a strongly correlated Gaussian), it could learn to use smaller ϵ and larger L in an adaptive fashion. In fact, the No-U Turn Sampler (NUTS; Homan and Gelman, 2014) is an adaptive method that explores this idea. L2HMC generalizes even further by instead modifying the Leapfrog integration steps (instead of their parameters only).

The suggested algorithm transforms the original Leapfrog equations

$$v^{1/2} = v - \tfrac{\epsilon}{2}\partial_x U(x); \qquad x' = x + \epsilon v^{1/2}; \qquad v' = v - \tfrac{\epsilon}{2}\partial_x U(x') \tag{1}$$

by adding scaling and translation factors:

$$\begin{aligned}
v' &= v \odot \overbrace{\exp\left(\tfrac{\epsilon}{2} S_v(\zeta_1)\right)}^{\text{Momentum scaling}} - \tfrac{\epsilon}{2}\Big[\partial_x U(x) \odot \overbrace{\exp\left(\epsilon Q_v(\zeta_1)\right)}^{\text{Gradient scaling}} + \overbrace{T_v(\zeta_1)}^{\text{Translation}}\Big] \\
x' &= x_{\bar{m}^t} + m^t \odot \Big[x \odot \exp\left(\epsilon S_x(\zeta_2)\right) + \epsilon\big(v' \odot \exp\left(\epsilon Q_x(\zeta_2)\right) + T_x(\zeta_2)\big)\Big] \\
x'' &= x'_{m^t} + \bar{m}^t \odot \Big[x' \odot \exp\left(\epsilon S_x(\zeta_3)\right) + \epsilon\big(v' \odot \exp\left(\epsilon Q_x(\zeta_3)\right) + T_x(\zeta_3)\big)\Big] \\
v'' &= v' \odot \exp\left(\tfrac{\epsilon}{2} S_v(\zeta_4)\right) - \tfrac{\epsilon}{2}\Big[\partial_x U(x'') \odot \exp\left(\epsilon Q_v(\zeta_4)\right) + T_v(\zeta_4)\Big].
\end{aligned} \tag{2}$$

These equations require a bit of explaining. First, each update is generalized by scaling the previous value (v or x), and also scaling and translating the updating value ($\partial_x U(x)$ or x). If we take the first equation, the original paper provides a short justification for this choice of update rule by saying that scaling the momentum can enable acceleration in low-density zones, facilitating mixing between

modes. It also mentions that scaling the gradient of the energy may allow better conditioning of the energy landscape (*e.g.*, by learning a diagonal inertia tensor, as we will see in one of the experiments). No reasons are given however regarding the use of the translation factor (was it better empirically?). As for the x update rules, most likely, the authors decided to use a similar looking update for aesthetic and uniformity reasons.

Second, these scaling and translating factors are the outputs of neural networks S, Q, and T, which take as input a subset ζ of the full state, in order to keep the Jacobian of the translation efficiently computable—this is because its determinant appears in the acceptance probability of Metropolis–Hastings. These subsets are defined as

$$\zeta_1 \triangleq \big(x, \partial_x U(x), t\big); \quad \zeta_2 \triangleq \big(x_{\bar{m}^t}, v, t\big); \quad \zeta_3 \triangleq \big(x'_{\bar{m}^t}, v, t\big); \quad \zeta_4 \triangleq \big(x'', \partial_x U(x''), t\big). \tag{3}$$

Third, keeping the Jacobian of the x update efficiently computable (independant of x) requires breaking up the update into two parts, following Real-valued Non-Volume Preserving transformations (RealNVP; Dinh et al., 2016). The mask m_t is drawn uniformly from the set of binary vectors half of whose entries are 0 and the other half being 1. We also wrote $\bar{m}^t = \mathbf{1} - m^t$ and $x_{m^t} = x \odot m^t$ for convenience of notation.

Metropolis–Hastings step. These steps now form our operator that is used for the proposition of each MH step. For MH to be well-defined however, this deterministic operator must be invertible and have a tractable Jacobian, *i.e.* we can compute its determinant. To make this operator invertible, the authors augment the state-space (x, v) into (x, v, d) where $d \in \{-1, 1\}$ is drawn with equal probability, and represents the direction of the updates. Equations (2) above represent the forward direction ($d = 1$). For the backward direction, we simply have to reverse the order of the updates (first take the mapping $v'' \to v'$, then $x'' \to x'$, followed by $x' \to x$ and finally $v' \to v$). These are explicitly given in the appendix of the original paper, alongside the derivation to compute the Jacobian for each update. For example, the Jacobian of the first update $v \to v'$ is $\exp\big(\frac{\epsilon}{2} S_v(\zeta_1)\big)$.

If we then define our Leapfrog operator to perform M steps of the above updates: $\mathbf{L}_\theta(x, v, d) = (x''_M, v''_M, d)$, and our flip operator as $\mathbf{F}(x, v, d) = (x, v, -d)$, we get our final invertible operator $\mathbf{FL}_\theta$ with tractable Jacobian $\mathcal{J}$. Using this operator, and defining $\xi = (x, v, d)$, the acceptance probability for the Metropolis–Hastings step is then found to be

$$A(\mathbf{FL}_\theta \xi \,|\, \xi) = \min\left(1, \frac{\pi(\mathbf{FL}_\theta \xi)}{\pi(\xi)} \, |\mathcal{J}(\mathbf{FL}_\theta)|\right). \tag{4}$$

Network architecture. All that is left is to define the different neural networks S_i, Q_i, and T_i (for $i \in \{x, v\}$) and craft a loss to train them. The architecture for x and v are the same so we will follow the paper and define S_v, Q_v, and T_v. First, the time step t is encoded as an harmonic signal $\tau(t) = (\cos(2\pi t/M), \sin(2\pi t/M))$ for reasons that are left unexplained. Then, using σ to denote ReLU, the 2-layer neural net is defined for n_h hidden units per layer as:

- $h_1 = \sigma(W_1 x + W_2 \partial_x U(x) + W_3 \tau(t) + b) \in \mathbb{R}^{n_h}$
- $h_2 = \sigma(W_4 h + b_4) \in \mathbb{R}^{n_h}$
- $S_v = \lambda_s \tanh(W_s h_2 + b_s); \quad Q_v = \lambda_q \tanh(W_q h_2 + b_q); \quad T_v = W_t h_2 + b_t$

where both λ_s and λ_q are learned parameters. The number of hidden units depends on the dimension of the distribution we are trying to sample from. The networks (see Figure 2) are trained with Adam (Kingma and Ba, 2014) and a learning rate of $\alpha = 10^{-3}$ for 5000 iterations with a batch size of 200.

Loss function and training. We must finally define some criterion to train the parameters θ of these networks. We want to learn MCMC kernels that converge fast and mix-in easily, both across energy levels and between modes. L2HMC's loss however is specifically targeted towards reducing burn-in and mixing time. The authors start by using a theorem from Pasarica et al. (2003) which shows that maximizing expected squared jumped distance (ESJD) is equivalent to minimizing lag-one autocorrelation. We know that the sum of the autocorrelation coefficients from $-\infty$ to ∞ is an estimate of the mixing time (Kipnis and Varadhan, 1986), so if we assume that lag-one autocorrelation trumps the other values of the sum, we can use maximizing ESJD as a proxy for minimizing mixing time. To be more formal, for ξ, ξ' in the extended state space, we define

$$\delta(\xi, \xi') = \delta\big((x, v, d), (x', v', d')\big) = \|x - x'\|_2^2. \tag{5}$$

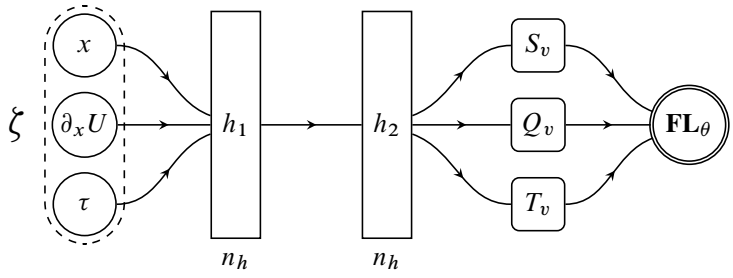

Figure 2: *Overview of network architecture for training S_v, Q_v, and T_v.* Time is initially mapped onto the unit circle to form a 3-tuple input $\zeta = (x, \partial_x U(x), \tau)$ with position and gradient information. The three networks are trained in parallel to speed up the computations. These form the basis of the new Leapfrog integrator $\mathbf{FL}_\theta$.

ESJD is thus $\mathbb{E}_{\xi\sim\pi(\xi)}[\delta(\mathbf{FL}_\theta\xi, \xi)A(\mathbf{FL}_\theta\xi \mid \xi)]$. The original work adds an extra reciprocal penalty for not moving enough. This new loss optimizes both for typical and worst case behaviour:

$$\ell_\lambda(\xi, \xi', A(\xi' \mid \xi)) = \underbrace{\frac{\lambda^2}{\delta(\xi, \xi')A(\xi' \mid \xi)}}_{\text{Stuck penalty}} - \underbrace{\frac{\delta(\xi, \xi')A(\xi' \mid \xi)}{\lambda^2}}_{\text{Large moves}}, \tag{6}$$

where λ is a scale parameter. In order to encourage both fast mixing time and burn-in time, the authors train their sampler by minimizing this loss over both the target distribution (mixing time) and initialization distribution (burn-in time). Formally, given a distribution π_0 over $\mathcal{X}$, we define $q(\xi) = \pi_0(x)\mathcal{N}(v; 0, I)\,p(d)$, and minimize

$$\mathcal{L}(\theta) \triangleq \underbrace{\mathbb{E}_{\pi(\xi)}\left[\ell_\lambda(\xi, \mathbf{FL}_\theta\xi, A(\mathbf{FL}_\theta\xi \mid \xi)\right]}_{\text{Fast mixing}} + \underbrace{\lambda_b\,\mathbb{E}_{q(\xi)}\left[\ell_\lambda(\xi, \mathbf{FL}_\theta\xi, A(\mathbf{FL}_\theta\xi \mid \xi))\right]}_{\text{Burn-in regularization}}. \tag{7}$$

Note that we can control how much we penalize long burn-in time using a user-defined scalar λ_b.

2 Experiments

2.1 Motivation for the experiments

The empirical evaluation of L2HMC is performed on a small collection of simple energy functions exhibiting common pathological behaviours in MCMC. Recall that L2HMC aims to learn a transition kernel with the following list of desiderata: fast mixing, fast burn-in, mixing across energy levels, and mixing between modes. To benchmark their algorithm, the authors train their model on four toy distributions that are known to cause problems even for a well-tuned HMC.

Ill-conditioned Gaussian (ICG): 50D Gaussian distribution with diagonal covariance matrix. The point of this experiment is to show that L2HMC can learn a diagonal inertia tensor. We follow the original paper in using diagonal values spaced log-linearly between 10^2 and 10^{-2}.

Strongly Correlated Gaussian (SCG): A diagonal 2D Gaussian with variances $[10^2, 10^{-2}]$ rotated by $\frac{\pi}{4}$. The authors claim that this problem shows that, although their parametric sampler only applies element-wise transformations, it can adapt to structure which is not axis-aligned. We fail to see how this is different from the ICG experiment (albeit being in two dimensions only).

Gaussian Mixture Model (GMM): Mixture of two isotropic Gaussians with $\sigma^2 = 0.1$ and centroids separated by distance 4. The means are thus about 12 standard deviations apart, making it almost impossible for HMC to mix between modes.

Rough Well (RW): Essentially a *wavy* Gaussian. Formally, for a given $\eta > 0, U(x) = \frac{1}{2}x^\top x + \eta\sum_i \cos(x_i/\eta)$. For small η the energy itself is altered negligibly, but its gradient is perturbed by a high frequency noise oscillating between -1 and 1. In the original experiment, the authors choose $\eta = 10^{-2}$ and we follow this convention.

To this set of four experiments from the original paper, we add a fifth of our choosing. Because L2HMC claims that it is an extension of HMC in that if the network sets $S = Q = T = 0$, we recover vanilla HMC, we wish to test whether the network will actually be able to learn to *not do anything* with a simple example:

Low Variance Gaussian (LVG): A simple 1D Gaussian distribution with variance $\frac{1}{25}$.

All experiments use $n_h = 10$ neurons per layer, except the 50D Gaussian where it is suggested to use $n_h = 100$. A step size of $\epsilon = 0.1$ and $M = 10$ Leapfrog steps were used for all experiments, as recommended by the authors.

2.2 Reproducing the main results

We confidently reproduced all of the four experiments from the original paper except for the ill-conditioned Gaussian, as can be seen by comparing Figure 3 and Table 1. To produce these plots after training, we generated 200 Markov chains for 2000 steps and computed the auto-correlation.

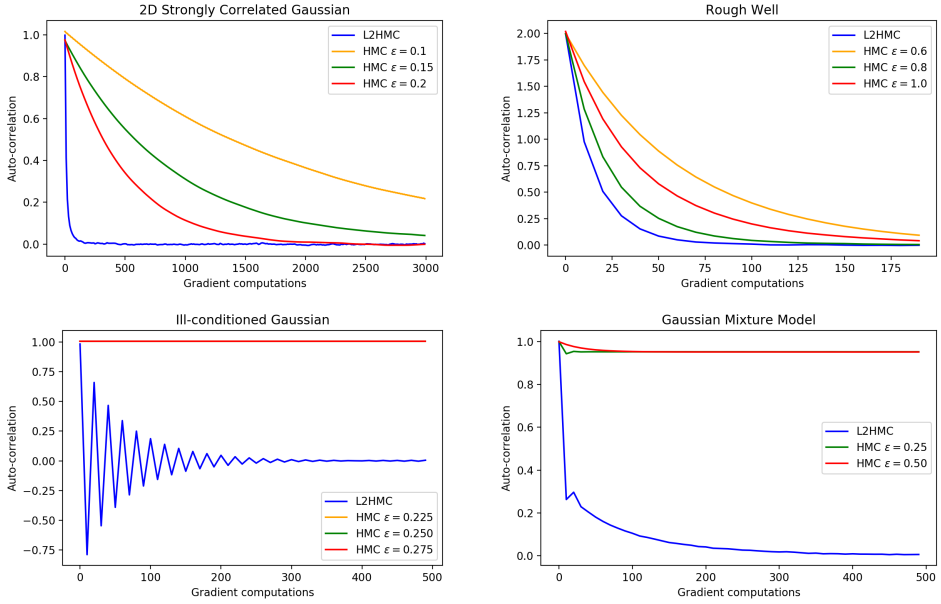

Figure 3: *Auto-correlation vs. gradient evaluations of energy* $U(x)$. All tasks but ICG agree with the original paper.

Distribution	ESS-L2HMC	ESS-HMC	Ratio
ICG	1.80×10^{-1}	2.49×10^{-4}	722
RW	1.92×10^{-1}	4.16×10^{-2}	4.6
SCG	3.29×10^{-1}	6.09×10^{-3}	53
GMM	1.77×10^{-1}	2.63×10^{-4}	674.6

Table 1: Effective sample size (ESS) per Metropolis–Hastings step.

3 Discussion

3.1 Analysis and comparison with original work

The strongly correlated Gaussian and the Rough well experiments align with the results obtained in the original paper. We report a ratio of 53 for SCG, while the authors reported 36.6. Similarly, the

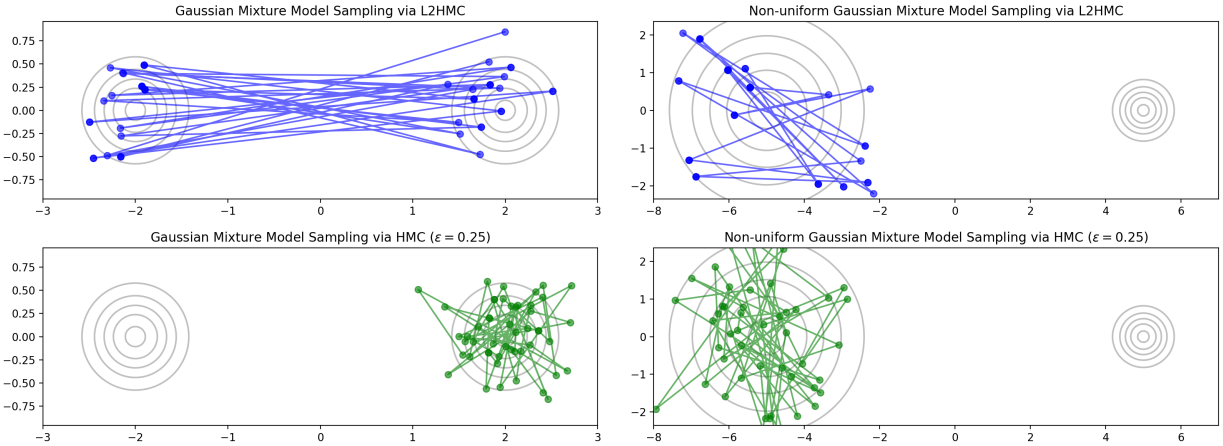

Figure 4: *Comparing L2HMC and HMC samples on a different mixtures of Gaussians.* L2HMC fails to explore the two modes when the variances are different, contradicting the claim made in (Lévy et al., 2018).

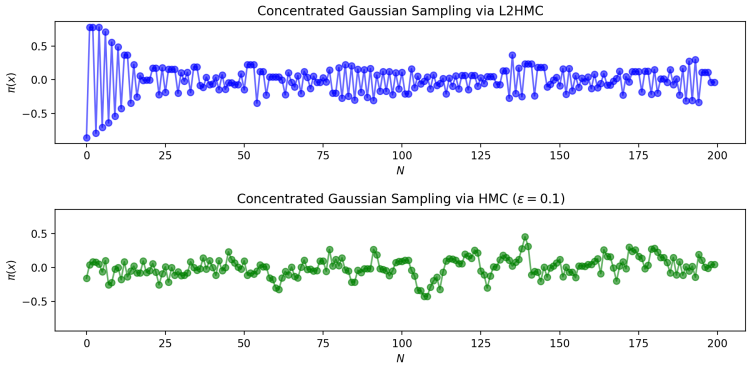

Figure 5: *Comparing L2HMC and HMC samples on our additional test, the low variance 1D Gaussian.* The former takes longer to burn-in but both explore the typical set properly.

RW has a ratio of 4.6, compared to 5.4. Since both values are in the same ballpark, **SCG and RW are both reproducible.**

For the ICG, we can see that using, as the authors did, diagonal values spaced log-linearly between 10^2 and 10^{-2}, we are not able to make the vanilla HMC auto-correlation values shrink to zero. However, we see an oscillatory motion in the L2HMC graph which is not observed in the original paper. Its auto-correlation value does shrink down to zero but it does so oscillating between positive and negative values. This can be explained by the model encouraging long jumps and hence forcing values to jump back and forth between two sides of a hyperplane centered at zero (imagine samples looking like a bipartite graph). This results in having auto-correlation values jumping back and forth between negative and positive values. Since we were unable to make the vanilla HMC models converge, the ESS ratio in Table 1 is far too high. In order to get the same results they got, we need to space the diagonal variance values log-linearly between 10^2 and 10^{-1}, and to reduce the vanilla HMC ϵ step-size to lower values. We managed to get $\epsilon = 0.1$ to converge, but did not try other values. Therefore, as far as our experiments show, **ICG is not reproducible.**

The other claim of the authors that we were not able to reproduce is that of the non-uniform mixture of Gaussians (MoG with very different covariance matrices). As can be seen in Figure 2.2, with equivariance Gaussians, L2HMC mixes well across modes, something HMC fails miserably at. For the non-uniform Gaussian case however, the authors compare their L2HMC to A-NICE-MC (Song et al., 2017), writing that because A-NICE-MC restricts itself to a volume-preserving operator, it is

unable to mix well between the two modes. This narrative suggests that only a fraction of the volume of the large mode can be mapped to the small one, making the L2HMC sampler outperform it. This is a *nice* claim, but we did not manage to verify it through our experiments. As such, we conclude that **GMM is partially reproducible.**

The last experiment we need to discuss is that of the 1D Gaussian that we added (LVG). We can see in Figure 5 that qualitatively speaking, HMC seems to outperform L2HMC. The samples from L2HMC have a high lag-one auto-correlation, as can be seen from them jumping back and forth. We see this as an impediment to the model. Indeed, in simple cases like this where it should technically learn to mimic HMC, it seems unable to do so, contrary to the theoretical claims of the authors.

3.2 Criticism

Construction of FL_θ. If we look back at Equations (2), we note that nowhere in the paper is the reason for keeping the ϵ parameter from HMC given: why is ϵ still required if we are learning scaling parameters? Couldn't the network learn these by itself? Furthermore, if we accept ϵ as necessary, then in the first and fourth updates why are the two scaling factors not multiplied by the same scalar: S_v is multiplied by $\frac{\epsilon}{2}$ whereas Q_v is multiplied by ϵ. Additionally, the authors do not motivate the use of the exponential function to multiply the scaling factors. This could have been added as the last layer of the network (*e.g.* just like a softmax), making the notation more readable.

Temperature annealing. One important detail missing from the paper is how they managed to achieve good results on the Gaussian mixture model example. If tried as is, the chain in the training phase will fail to find the second mode (as can be seen from the Vanilla HMC sample in Figure 2.2), and hence the network will not be able to learn the jump between the modes. In order for the network to achieve this hidden mode exploration, the chain needs to find the second mode *in the first place*. This, we achieved by adding temperature annealing to the training phase. By starting with a high temperature $T = 15$, the chain is able to move between both modes. Once it has learned this, we can lower the temperature back to $T = 1$, hence recovering the initial distribution while keeping the jump in the network's "memory". This precious detail was missing from the original paper and was obtained by communicating with Daniel Lévy directly.

Burn-in regularization. We are puzzled by the justification for the loss function. Why does the second term of the loss encourage fast burn-in? ESJD is only equivalent to fast mixing, so it seems like the second term encourages fast mixing starting with the initial distribution, which is not necessarily equivalent to fast burn-in time.

Auto-correlation plot for GMM. Another mistake the authors make can be found in the auto-correlation graph (3) for this Gaussian Mixture Model. They find a constant auto-correlation close to 1 for the vanilla HMC models but this is only because of an error in their code. The base code forgets to subtract the mean from the samples, and because the HMC model does not mix between the two modes, all of the samples have the same sign for x. Hence of course they seem correlated if we forget to subtract the mean. We kept the code the same to show their mistake in this graph.

Additional minor typos. One small detail that should be fixed in the paper is that of the definition of their neural network in the appendix. Q_v, S_v, and T_v have x, $\partial_x U$, and τ as input (see our Figure 2), and not x, v and τ as the authors write.

4 Conclusion

In summary, our top 3 criticisms regarding L2HMC are

1. Lack of transparency regarding how and why temperature annealing should be used for better mixing between modes;
2. Lack of motivation regarding the design choices for the modified Leapfrog integrator;
3. Lack of details regarding the loss function, especially the term related to burn-in regularization.

We need to mention that we did not attempt to reproduce the VAE part of the paper. This secondary experiment is merely an application of L2HMC to latent-variable generative models and did not offer, in our eyes, any theoretical insights on the method that was worth the time.

To conclude, we do think that the ideas presented in Lévy et al. (2018) are novel and promising. Applying deep learning techniques to Bayesian inference is definitely an avenue that will continue to flourish in the near future. We remain doubtful, however, as to how L2HMC could scale to more complex analytical distributions given that first-order gradient computations can incur a nonneglible cost to the training procedure.

References

Michael Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Estimation using Real NVP. *CoRR*, abs/1605.08803, 2016.

Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216,222, 1987.

Stuart Geman and Donald Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(6):721–741, November 1984. ISSN 0162-8828.

Matthew D. Homan and Andrew Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, January 2014.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

C. Kipnis and S. R. S. Varadhan. Central Limit Theorem for Additive Functionals of Reversible Markov Processes and Applications to Simple Exclusions. *Communications in Mathematical Physics*, 104(1):1–19, 1986.

Daniel Lévy, Matt D. Hoffman, and Jascha Sohl-Dickstein. Generalizing Hamiltonian Monte Carlo with Neural Networks. In *International Conference on Learning Representations*, 2018.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087,1092, 1953.

Radford M. Neal et al. MCMC using Hamiltonian Dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11), 2011.

Cristian Pasarica, Andrew Gelman, and J. P. Morgan. Adaptively Scaling the Metropolis Algorithm using Expected Squared Jumped Distance. Technical report, Columbia University, 2003.

Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-NICE-MC: Adversarial Training for MCMC. *arXiv preprint arXiv:1706.07561*, 2017.